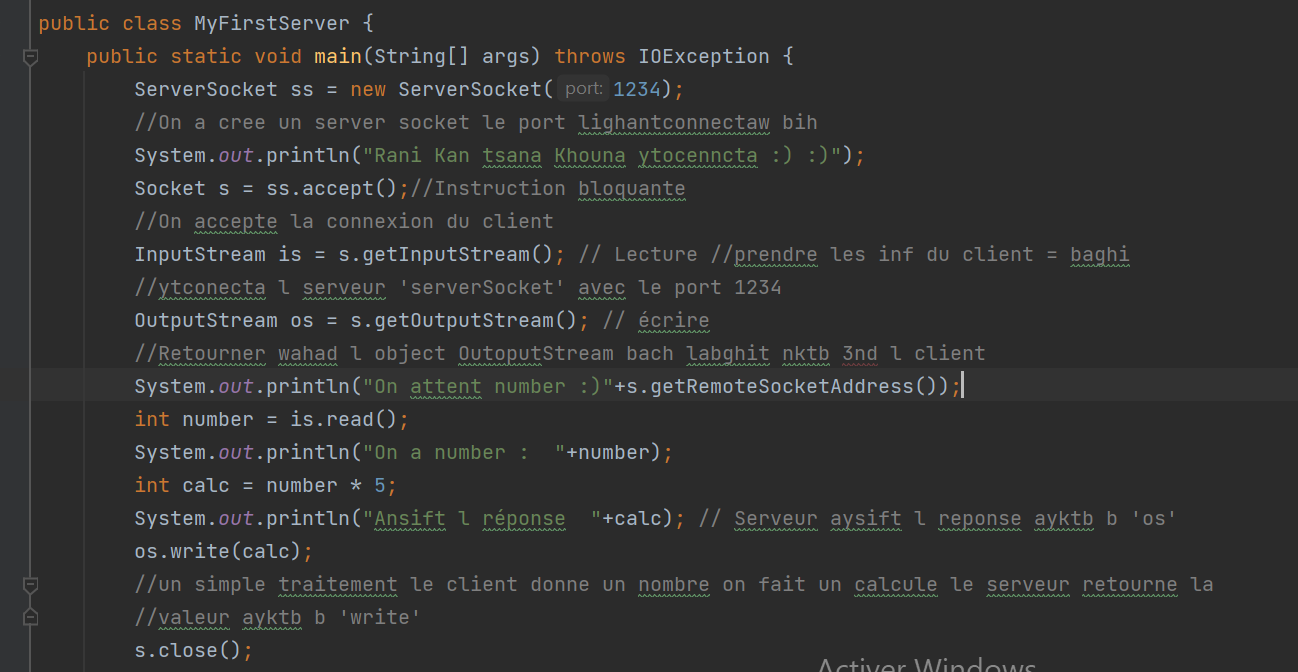
Rapport Application Client Serveur

Réalisé par : Darfaoui Fadwa

Ouarray Fatimaezzahra

* On va créer une application client serveur



On a créer la classe « MyFirstServer » :

ServerSocket ss = new ServerSocket( 1234 ) ; => Créer un object server socket qui écoute sur le port 1234. Cela signifie que le serveur sera disponible pour les connexion entrantes sur ce port.

Socket s = ss.accept( ) ; => Attent qu’un client se connecte et accepte la connexion. Cette instruction est blocante, ce qui signifie que le programme attendra ici jusqu’à ce qu’un client se connecte.

InputStream is = s.getInputStream(); => récupère un flux d'entrée à partir du socket du client. Cela permettra au serveur de lire les données envoyées par le client.

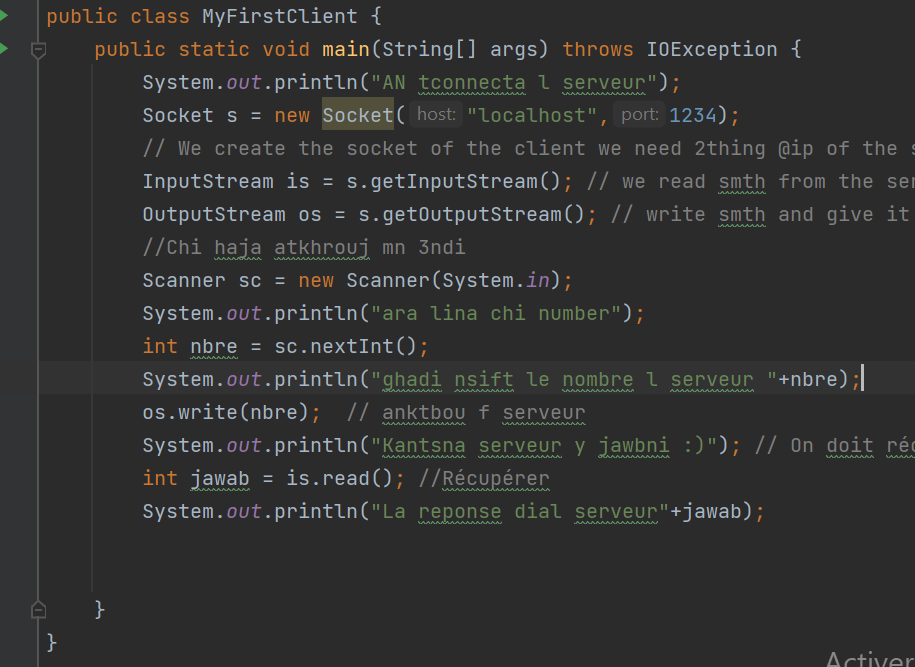
OutputStream os = s.getOutputStream();=> récupère un flux de sortie à partir du socket du client. Cela permettra au serveur d'envoyer des données au client.

int number = is.read();=> lit un entier envoyé par le client à partir du flux d'entrée. Cette instruction est bloquante jusqu'à ce qu'un entier soit reçu.

int calc = number \* 5;=> effectue un calcul simple en multipliant le nombre reçu par 5.

os.write(calc); =>envoie la réponse calculée au client via le flux de sortie.

s.close(); =>ferme la connexion avec le client.



Dans La classe MyFirstClient :

Socket s = new Socket("localhost", 1234); => crée un objet Socket pour établir une connexion avec le serveur. Le premier argument "localhost" indique l'adresse IP du serveur, dans ce cas, il s'agit de la même machine locale. Le deuxième argument 1234 spécifie le port sur lequel le serveur écoute.

InputStream is = s.getInputStream();=> récupère un flux d'entrée à partir du socket du serveur. Cela permettra au client de lire les données envoyées par le serveur.

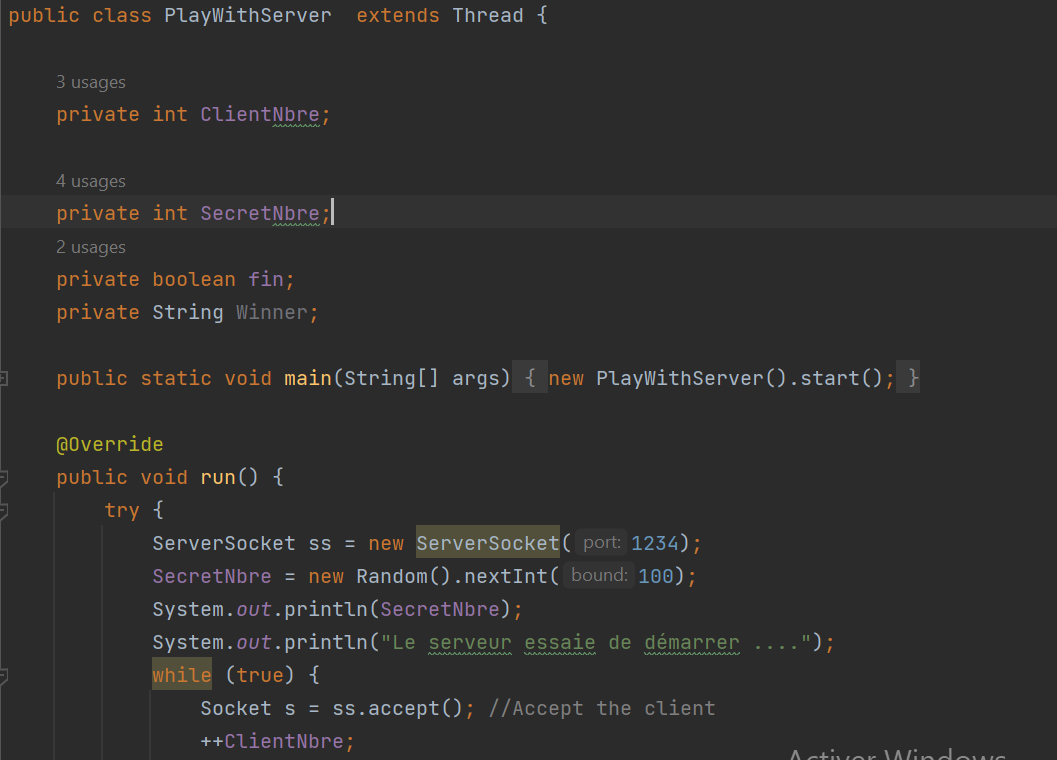
OutputStream os = s.getOutputStream(); => récupère un flux de sortie à partir du socket du serveur. Cela permettra au client d'envoyer des données au serveur.

Scanner sc = new Scanner(System.in);=> crée un objet Scanner pour lire les entrées de l'utilisateur à partir de la console.

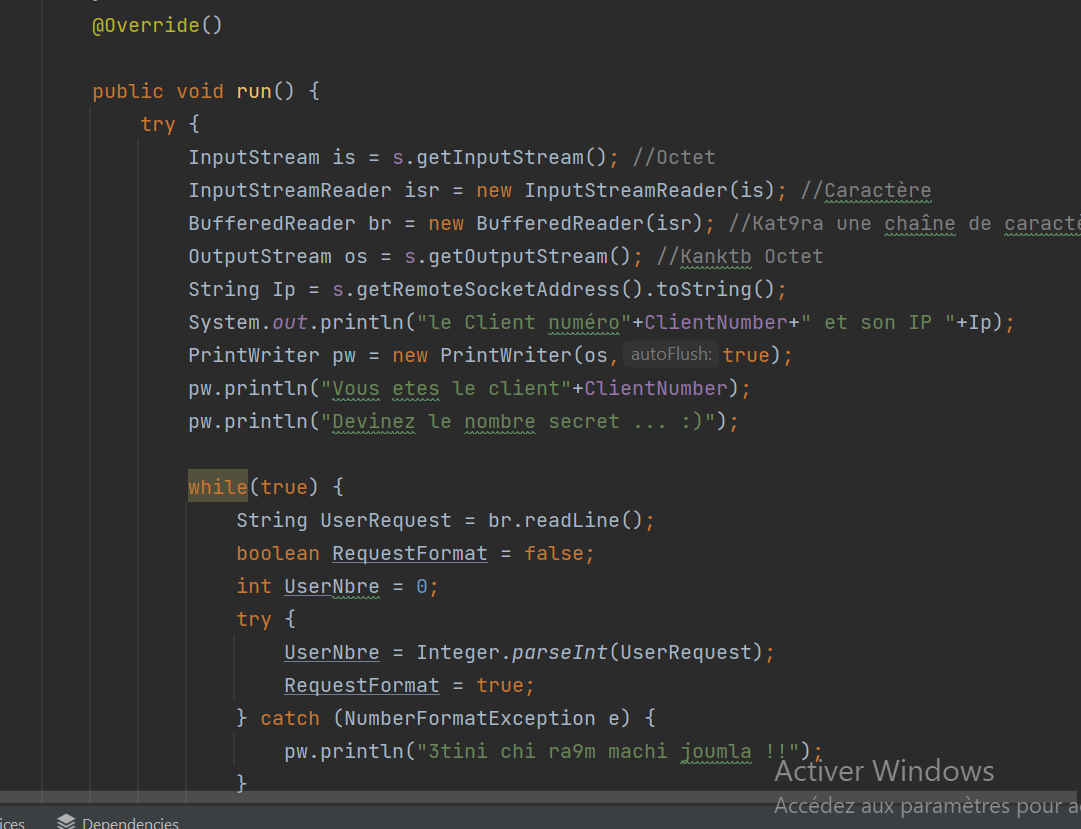
int nbre = sc.nextInt(); => lit un entier entré par l'utilisateur à partir de la console.

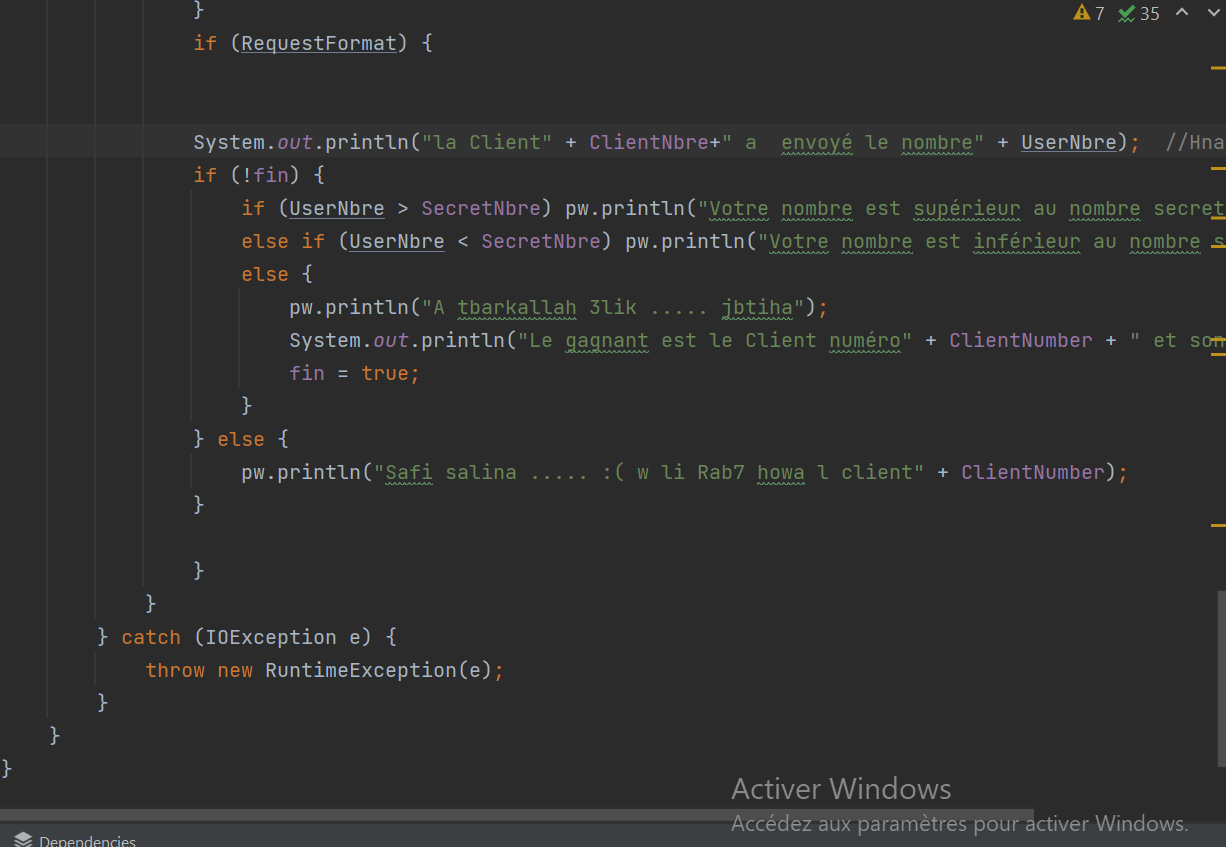
os.write(nbre);=> envoie le nombre au serveur via le flux de sortie.

int jawab = is.read(); => lit la réponse envoyée par le serveur à partir du flux d'entrée. Cette instruction est bloquante jusqu'à ce qu'une réponse soit reçue.









Dans la classe « PlayWithServer » :

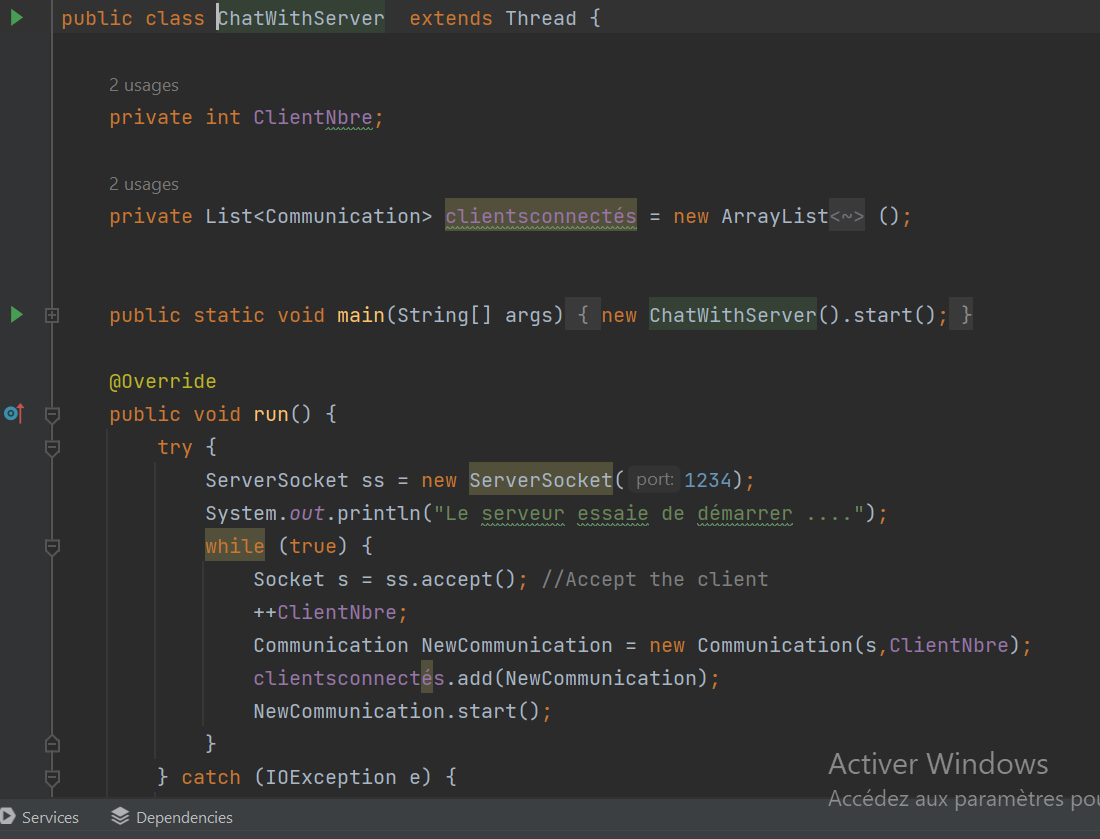
La classe PlayWithServer hérite de la classe Thread, ce qui signifie qu'elle peut être exécutée en tant que thread séparé. Et La méthode main est la méthode principale du programme. Elle crée une instance de PlayWithServer et la démarre en tant que thread.

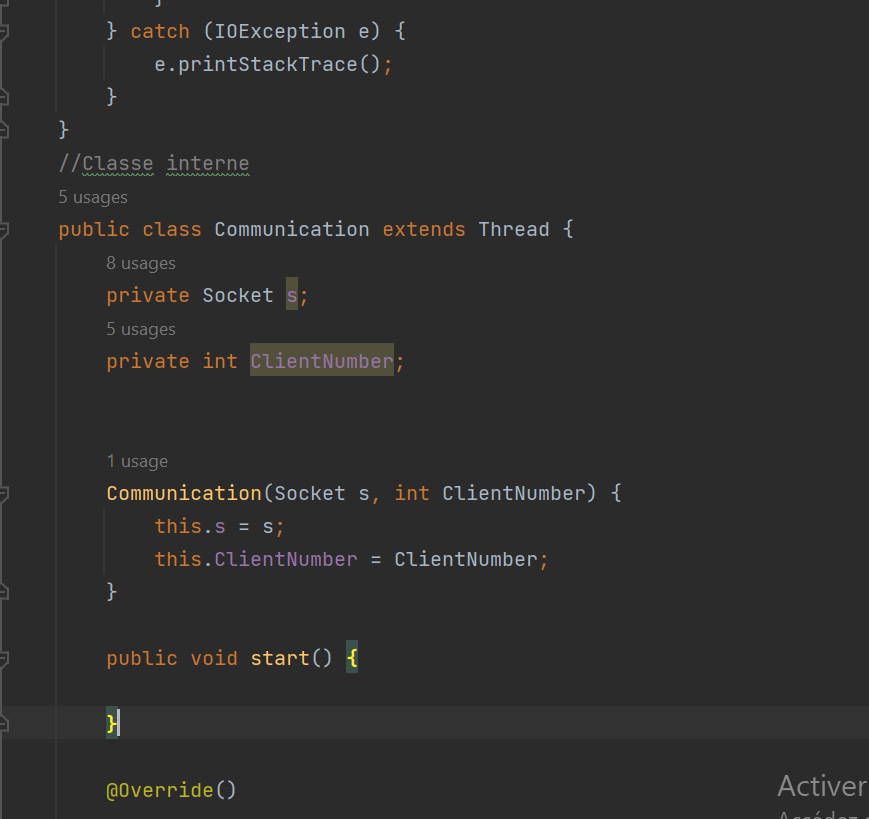
‘’La méthode run’’ est une méthode héritée de la classe Thread qui contient le code exécuté par le thread. Elle crée un ServerSocket qui écoute sur le port 1234. Un nombre secret est généré aléatoirement. Ensuite, une boucle infinie accepte les connexions des clients et crée un thread de communication pour chaque client.

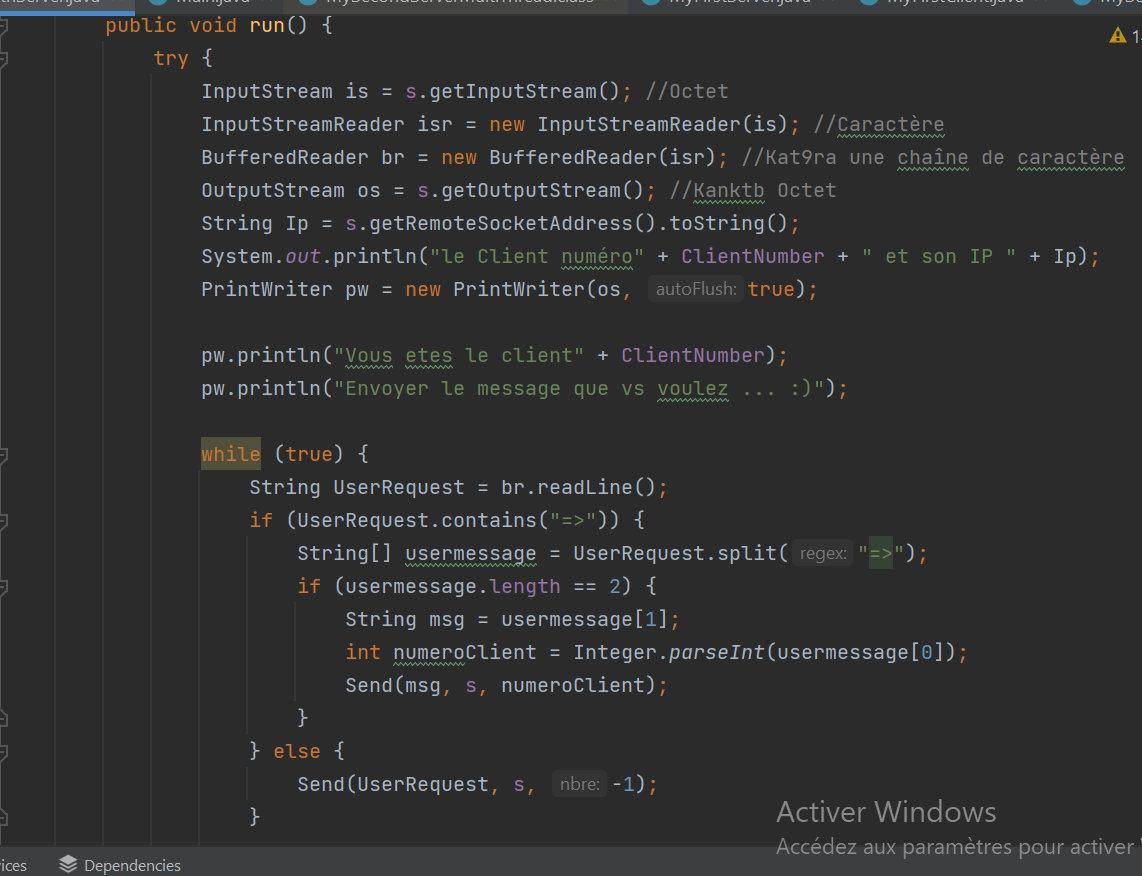
‘’La classe interne Communication’’ étend également la classe Thread et gère la communication entre le serveur et un client spécifique.

‘’La méthode run’’ récupère les flux d'entrée et de sortie du client, demande au client de deviner le nombre, et Une boucle infinie commence pour lire les réponses du client.

La variable ‘’ UserRequest ‘’ lit la réponse du client à partir du flux d'entrée. La variable RequestFormat est utilisée pour vérifier si la réponse est un nombre valide. Si la réponse n'est pas un nombre valide, un message d'erreur est envoyé au client.







La Classe « ChatWithServer »  :

La liste clientsconnectés est utilisée pour stocker les objets Communication représentant les connexions clients.

La méthode main instancie et démarre une nouvelle instance de ChatWithServer pour lancer le serveur.

Dans la méthode run, un ServerSocket est créé et lié au port 1234. Ensuite, le serveur entre dans une boucle infinie où il attend les connexions des clients. Lorsqu'un client se connecte, un objet Communication est créé pour gérer la communication avec ce client, et cet objet est ajouté à la liste clientsconnectés.

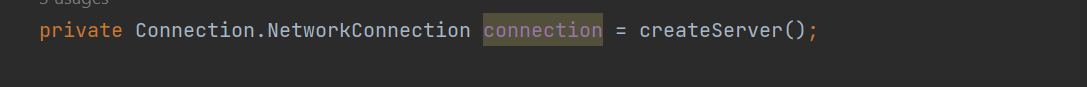
La classe interne Communication gère la communication avec un client.

La méthode run est l'endroit principal où la communication avec le client est gérée. Elle récupère les flux d'entrée et de sortie du socket pour lire les messages du client et envoyer des réponses.

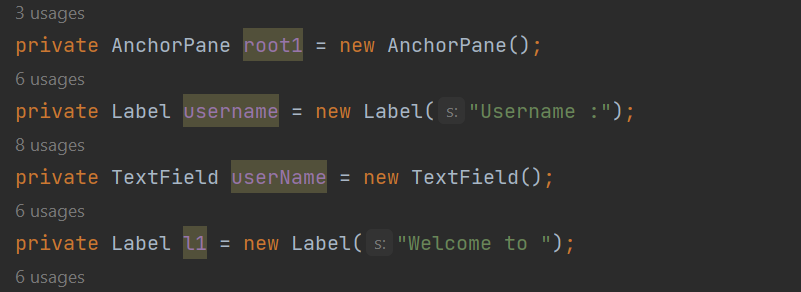
Un message est envoyé au client avec son numéro attribué. Ensuite, le serveur attend les messages du client dans une boucle infinie. Les messages entrants sont lus à l'aide d'un BufferedReader à partir du flux d'entrée. Si le message contient la séquence "=>", cela signifie qu'il s'agit d'un message destiné à un client spécifique. Le message est alors analysé pour extraire le numéro du client et le contenu du message, puis la méthode Send est appelée pour envoyer ce message au client.

La méthode Send parcourt la liste des clients connectés et envoie le message à tous les clients, sauf à l'expéditeur du message initial est spécifié est -1, cela signifie que le message est destiné à tous les clients.

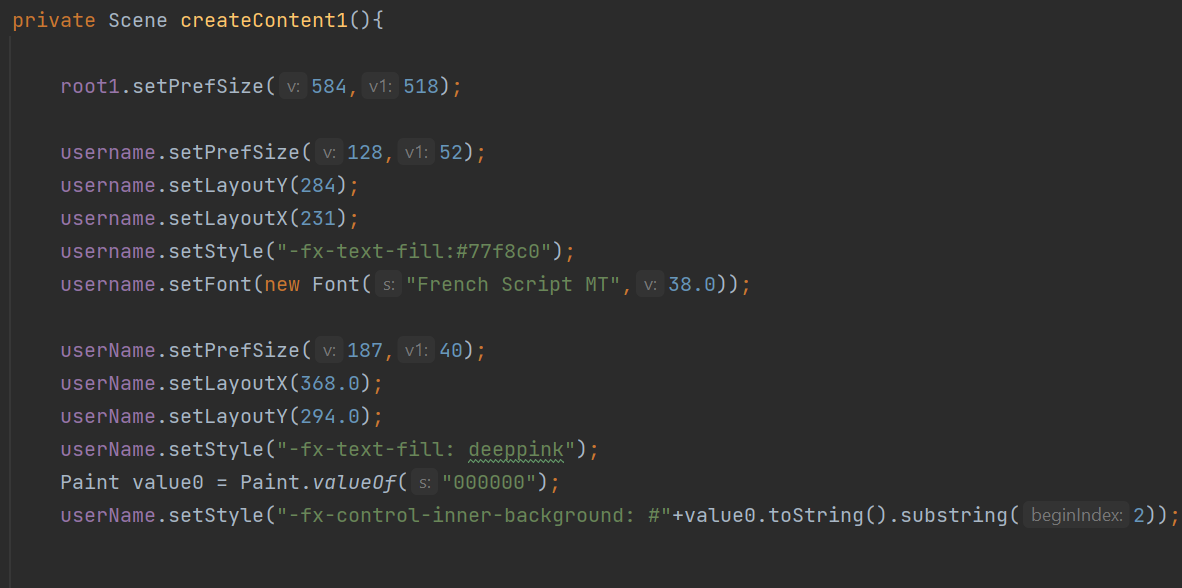
# L’Interface Aplication Client Serveur :



La variable `connection` de type `Connection.NetworkConnection` en utilisant la fonction `createServer()`, qui retourne une instance de connexion.



Ces variables sont utilisées pour créer des éléments d’interface utilisateur , où root1 est un conteneur principal, username est une étiquette affichant le texte “Username :”, et userName est un champ de saisie de texte permettant à l’utilisateur de saisir un nom d’utilisateur.



La fonction `createContent1()` crée et retourne un objet de type `Scene` qui représente le contenu graphique d'une application JavaFX. Elle configure les éléments graphiques tels que des conteneurs, des boutons, des étiquettes, etc., en fonction des besoins de l'interface utilisateur. Une fois les éléments graphiques assemblés, la scène est retournée et peut être utilisée pour afficher l'interface utilisateur correspondante.

root1.setPrefSize(584,518);`: Définit la taille préférée du conteneur `root1` à 584 pixels de largeur et 518 pixels de hauteur.

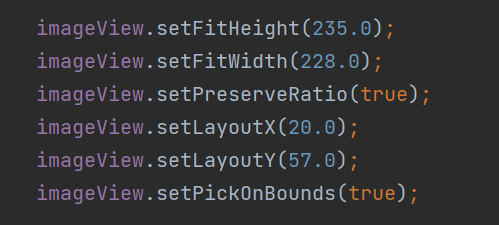
username.setPrefSize(128,52); : Définit la taille préférée de l'étiquette `username` à 128 pixels de largeur et 52 pixels de hauteur.

username.setLayoutY(284); : Définit la position verticale de l'étiquette `username` à 284 pixels par rapport à l'origine de son conteneur parent.

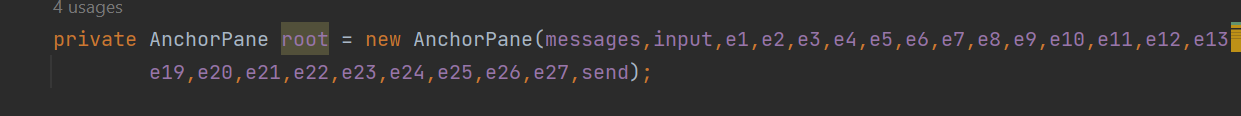
username.setLayoutX(231); : Définit la position horizontale de l'étiquette `username` à 231 pixels par rapport à l'origine de son conteneur parent.

username.setStyle("-fx-text-fill:#77f8c0");` : Applique un style CSS à l'étiquette `username`, en définissant la couleur du texte sur une teinte de vert.

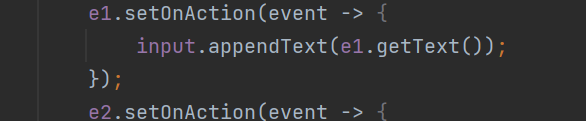
username.setFont(new Font("French Script MT",38.0));` : Définit la police de caractères de l'étiquette `username` sur "French Script MT" avec une taille de police de 38 points.



Ici, On définit les propriétés visuelles et la disposition d’un objet imageView.



La variable root de type AnchorPane qui représente un conteneur graphique dans une application JavaFX. Le AnchorPane est initialisé avec plusieurs éléments, tels que messages, input, e1, e2, e3, …, e27, et send.



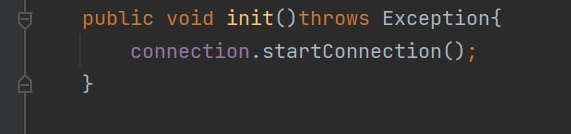
Lorsque l’événement associé à e1 se produit (par exemple, lorsqu’il est cliqué), la fonction lambda event -> { input.appendText(e1.getText()); } est exécutée.



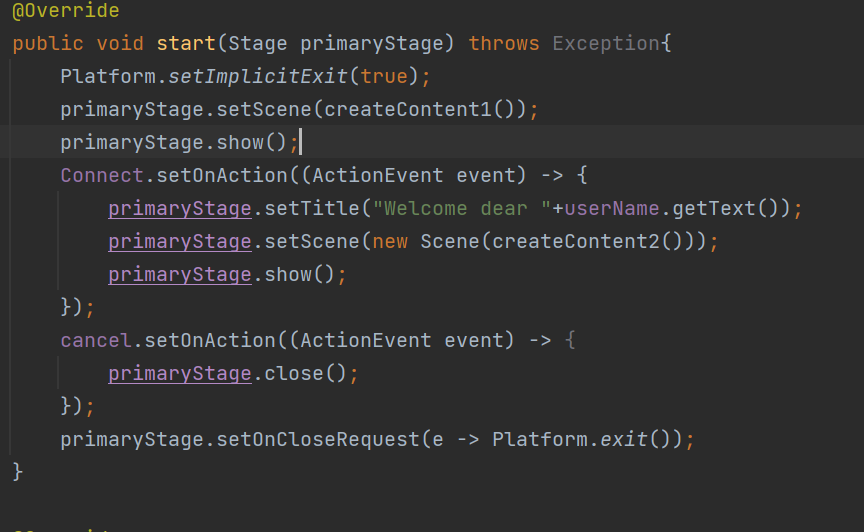
On désactive la possibilité d’éditer le composant messages.

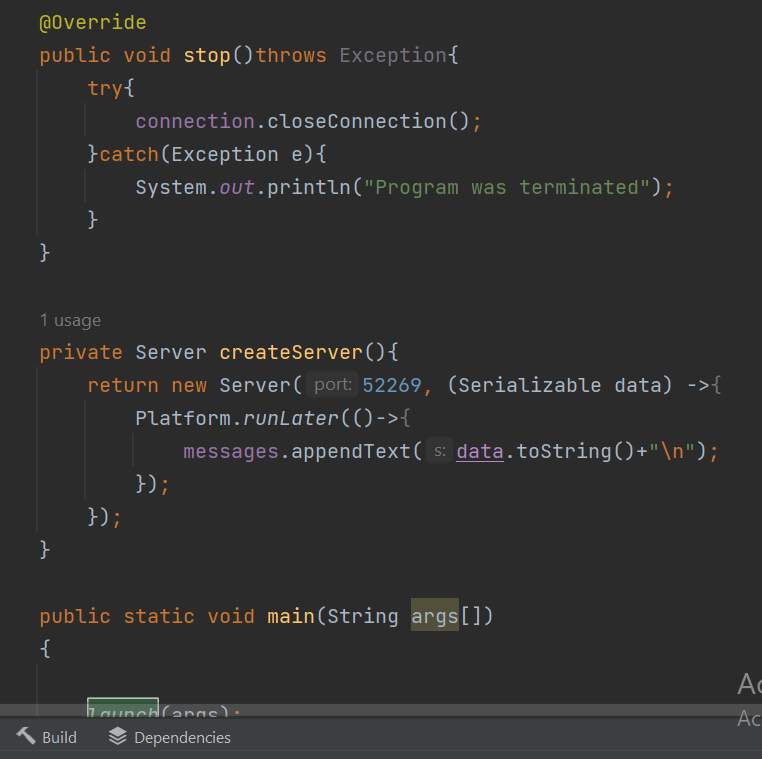


cette action est déclenchée lorsque le bouton send est cliqué. Elle récupère le texte saisi par l’utilisateur, l’envoie via la connexion et affiche le message correspondant dans la zone de texte messages.



la méthode init() ,elle appelle la méthode startConnection() pour établir la connexion .





Ici on a des actions configurées pour les boutons Connect et cancel, une initialisation de la fenêtre principale, une méthode de fermeture de l’application, la création d’un serveur et la définition des actions à effectuer lorsque la fenêtre principale est fermée.